CSC249/449 HW3, Kefu Zhu

Set Up

- **pytorch version**: 1.0.1.post2

CNN Feature Extractor

Parameters

- **batch size**: 4
- **learning rate**: 0.001
- **optimization method**: Stochastic gradient descent (SGD) with momentum
    - **momentum**: 0.9
- **epoch number**: 6

Performance

	Accuracy
Overall	**64%**
plane	75%
car	88%
bird	46%
cat	39%
deer	66%
dog	49%
frog	69%
horse	67%
ship	67%
truck	75%

DCFNET Tracker

Performance

Loading param.pth	Loading classifier_param.pth	accumulate w	accumulate x	result (AUC)
✓	×	✓	✓	0.6350
×	✓	✓	✓	0.5232

Mechanism of the CNN Feature Extractor

The feature extractor is trained on Convolutional Neural Network (CNN) which is a specific neural network that works well with image recognition. Each iteration in the training of CNN includes two main stage: feed-forward and back-propagation

feed-forward

1. Given an image, in the first layer, CNN will perform convolution operation on the image using kernels. User can define the following paramters

    - The number of kernels
    - The value for padding
    - The value for stride

2. After the first layer, user can choose add an activation layer on top. In this assignment, we used *Rectified Linear Units (ReLu)* as our activation function

3. After the convolution operation, user can also choose to perform downsampling by adding a pooling layer after the first layer. There are a number of choices available for pooling layer. In this assignment, we used *Max Pooling* after the *ReLu* activation layer

4. To extract higher level features, user can choose to add more layers to the convolutional neural network. In this assignment, we used

    - 4 convolutional layers
    - 4 *ReLu* activation layers
    - 2 *Max Pooling* layers

5. After the image processing through multiple convolutional + pooling layers, we map the extracted features with target classes using fully connected layers. In this assignment, we used 3 fully connected layers along with *ReLu* activations.

    - 3 fully-connected layers (*ReLu* activation)

back-propagation

Since we have our prediction for a given image after the feed-forward, we can calculate our loss and back propagate our loss to each layer. Many loss functions and optimization methods existed. In this assignment, we used *Cross Entropy* as our loss function and update the parameters using *Stochastic Gradient Descent (SGD) with Momentum*

Cross-Entropy

$$Loss = -\sum_{c=1}^{M} y_{o,c} \cdot log(p_{o,c})$$

SGD with momentum

$$V_t = \alpha \cdot V_{t-1} + (1-\alpha) \cdot \nabla_w Loss$$

$$W = W - \eta V_t$$

By iterate this mechanism over and over, the convolutional neural network will be able to learn to extract features that are more and more useful in predicting image correctly based on the labels provided in the training dataset.

Mechanism of DCFNET Tracker

For a given video, at time frame $t = 1$, we first initialize our target patch and then crop it into a patch x. We then extract features from the patch x and obtain $\phi(x)$, using the pre-trained convolutional neural network.

For the feature-extracted patch $\phi(x)$, we try to learn an optimal filter w that will gives us a gaussian y with peak in the center.

In the next time frame $t = 2$, we crop a new search patch z that is centered at the same position with x. Similarly, we apply feature extraction using convolutional neural network on z and obtain $\phi(z)$. We then perform cross correlation on $\phi(z)$ using the filter w we learned in the previous time frame, which is a representation of the target patch in $t = 1$.

If we denote the result of the cross correlation as g, the response map, we can then track the position of the target in the new time frame, $t = 2$, by seaching the maximum value (peak) of the response map g. We then update the location of our tracking box and the correlation kernel w, using the new maximum value.

We can repeat such procedure for the following time frame $t = 3, 4, 5, \ldots$ By doing so, we will be able to track the movement of the target in video for each time frame.